**Q1. Explain the difference between greedy and non-greedy syntax with visual terms in as few words as possible. What is the bare minimum effort required to transform a greedy pattern into a non-greedy one? What characters or characters can you introduce or change?**

**ANS:-**

**Greedy version, Python matches the longest possible string**

**import re**

**text = "<Robot is the latest addition to the tech items> <Robot is very advanced> <Robot is a machine>"**

**greedyregobj=re.compile(r'<.\*>')**

**match=greedyregobj.search(text)**

**print(match.group())**

**the Non-greedy version of the regex, Python matches the shortest possible string**

**nongreedyregobj=re.compile(r'<Ro.\*?>')**

**match1=nongreedyregobj.search(text)**

**match1.group()**

**<Robot is the latest addition to the tech items> <Robot is very advanced> <Robot is a machine>**

**'<Robot is the latest addition to the tech items>'**

**Q2. When exactly does greedy versus non-greedy make a difference?  What if you're looking for a non-greedy match but the only one available is greedy?**

**ANS:-**

***In the non-greedy version of the regex, Python matches the shortest possible string. In the greedy version,Python matches the longest possible string. If only non greedy match is available, we can use other filtering or pattern matching methods of regex and further identify the required pattern.***

**Q3. In a simple match of a string, which looks only for one match and does not do any replacement, is the use of a nontagged group likely to make any practical difference?**

**ANS:-**

**import re**

**phoneNumRegex = re.compile(r'\d\d\d')**

**mo = phoneNumRegex.search('My number is 415-555-4242.')**

**print('Phone number found: ' + mo.group()) *# non tagged group***

**print('Phone number found: ' + mo.group(0))**

**Phone number found: 415**

**Phone number found: 415**

**Q4. Describe a scenario in which using a nontagged category would have a significant impact on the program's outcomes.**

**ANS:-**

***Non tagged category :***

**import re**

**text='135.135'**

**pattern=r'(\d+)(?:.)(\d+)'**

**regobj=re.compile(pattern)**

**matobj=regobj.search(text)**

**matobj.groups()**

***# Here the '.' decimal is not tagged or captured.***

***# It will useful in scenarios where the separator of value in # a string is of no use and we need to capture only the***

***# values.***

**('135', '135')**

**Q5. Unlike a normal regex pattern, a look-ahead condition does not consume the characters it examines. Describe a situation in which this could make a difference in the results of your programme.**

**ANS:-**

***While counting the number of multiple lines or multiple sentence in a string the positive look ahead makes a difference, without which we won’t get the correct count of lines or sentences in a string.***

**Q6. In standard expressions, what is the difference between positive look-ahead and negative look-ahead?**

**ANS:-**

***Positive look ahead is an assertion continuing the search and extending the string e.g.pattern= r'abc(?=[A-Z])''.***

***Here after 'abc', ? is extending the search and says that in the remaining string, first identify the next character should be capitalized character between A and Z, but doesn’t consume it.***

***# Example of Positive lookahead***

**import re**

**pat=r'abc(?=[A-Z])'**

**text="abcABCEF"**

**regobj=re.compile(pat)**

**matobj=regobj.findall(text)**

**print("Positive lookahead:",matobj)**

**Positive lookahead: ['abc']**

***Negative look head is also an assertion to exclude certain patterns e.g. pattern = r'abc(?!abc)', means identify a substring containing 'abc' which is not followed by another 'abc'***

***# Example of Negative lookahead***

**import re**

**pat1=r'abc(?!abc)'**

**text1="aeiouabcabc"**

**regobj1=re.compile(pat1)**

**matobj1=regobj1.findall(text)**

**print("Negative look ahead:",matobj1)**

**Negative look ahead: ['abc']**

**Q7. What is the benefit of referring to groups by name rather than by number in a standard expression?**

**ANS:-**

***Ans : The benefit of referring to the groups by name is that***

***1) The code is clear***

***2) It is easier to maintain the code.***

**Q8. Can you identify repeated items within a target string using named groups, as in "The cow jumped over the moon"?**

**ANS:-**

**import re**

**text = "The cow jumped over the moon"**

**regobj=re.compile(r'(?P<w1>The)',re.I)**

**regobj.findall(text)**

**['The', 'the']**

**Q9. When parsing a string, what is at least one thing that the Scanner interface does for you that the re.findall feature does not?**

**ANS:-**

***Ans : re.search() method either returns None (if the pattern doesn’t match), or a re.MatchObject that contains* *information about the matching part of the string. This method stops after the first match, so this is best* *suited for testing a regular expression more than extracting data, whereas* *Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left to right, and matches are returned in the order found.***

**Q10. Does a scanner object have to be named scanner?**

**ANS:-**

***The scanner object need not be named scanner. It may have any name.***